

OpenShift Guide

Adrian Kosmaczewski

Version 1.0, 2023-10-13

Table of Contents

About the Author	1
1. Course Requirements	2
1.1. Audience	2
1.2. Recommended Tools	2
1.3. Knowledge	2
1.4. Example Code	3
1.5. Recommended LinkedIn Learning Courses	3
2. GitHub Repository	4
2.1. Branches	4
2.2. Beware!	4
Getting Started	6
3. What is OpenShift?	7
3.1. What is Red Hat OpenShift?	7
3.2. Is Red Hat OpenShift Open Source?	8
3.3. How can I run OpenShift?	8
4. OpenShift-only Custom Resource Definitions	9
4.1. Project	9
4.2. Route	9
4.3. DeploymentConfig	9
4.4. BuildConfig	10
5. OpenShift-only Command-Line Tools	11
5.1. oc	11
5.2. odo	11
6. Red Hat OpenShift Developer Sandbox	13
6.1. Demo	13
6.2. Dev Spaces	14
7. Red Hat OpenShift Local	15
7.1. Requirements	15
7.2. Hardware Requirements	16
7.3. Installation	16
Standard DevOps Practices	18
8. Understanding OpenShift Security	19
8.1. Users and Roles	19
8.2. Security Context Constraints	19
8.3. Base Images	19
8.4. Building Images	20
8.5. Network Ports	20
8.6. Storage	20

8.7. Apply "12 Factor" Best Practices	21
9. Deploying and Debugging Containers	22
9.1. Deploying Applications with the Web Console	22
9.2. Creating and Debugging Applications with the <code>odo</code> Tool	23
10. Building Containers from Source Code	25
10.1. Deploying	25
10.2. Container Registry	26
10.3. Updating the Application	27
11. Using CI/CD Pipelines	28
11.1. Install Tekton	28
11.2. Example	28
11.3. Inspecting the Pipeline	29
Advanced Cloud Native Apps	31
12. Templates and Operators	32
12.1. Templates	32
12.2. Operators	33
13. Serverless with Knative	34
14. Microservices and Service Mesh	36
14.1. Installation	36
14.2. Using Istio and Kiali	37
Scaling and Monitoring Apps	39
15. Logging	40
15.1. Checking Pod Logs on the Web Console	40
15.2. Using Elasticsearch and Kibana	40
16. Manual, Horizontal, and Vertical scaling	42
17. Monitoring apps with Prometheus	44
17.1. Instrumenting Applications	44
17.2. Observing Applications	45
Impressum	47
GDPR Disclaimer	47
Copyright	47
Toolchain	47
Index	48

About the Author

Adrian Kosmaczewski is a software expert with over 25 years of experience. He is a [published author](#), trainer, and speaker. He has written many books about software development and has shipped cloud, mobile, and desktop apps since 1996. Adrian holds a Master in Information Technology from the University of Liverpool.

Currently in charge of Developer Relations at [VSHN – The DevOps Company](#) at Zürich, Switzerland and Vancouver, BC, Canada. (VSHN is a [Red Hat Premier CCSP Business Partner](#), and features the highest quantity and quality of Red Hat OpenShift experts in Switzerland.)

Adrian is the co-creator of [De Programmatica Ipsum](#), a monthly publication about the impact of software in society.

When not coding or teaching, Adrian likes to spend time with his wife Claudia, his cat Max and his Olivetti Lettera 22 typewriter.

Chapter 1. Course Requirements

This section enumerates some knowledge and tools that would be useful to follow this course about Red Hat OpenShift.

1.1. Audience

This course is geared towards developers interested in running applications in OpenShift. It is also useful for DevOps engineers who want to learn how to use OpenShift to build and deploy cloud native applications.

1.2. Recommended Tools

Here's some basic tooling to try Red Hat OpenShift by yourself:

- Operating System
 - Microsoft Windows, Apple macOS, or Linux.
- Applications
 - Git
 - [Podman](#), [Podman Desktop](#), [Docker](#), or [Docker Desktop](#)
- Accounts
 - A [Red Hat Developer account](#)
 - A GitHub account

1.3. Knowledge

You should be familiar with:

- Containers
 - Dockerfile and Containerfile
 - Docker or Podman
 - Container registries
- Kubernetes
 - Deploying applications
 - Manifests in YAML format
 - Performing basic administration tasks
- Git
 - [GitHub](#) and [GitLab](#)

1.4. Example Code

This course uses 4 simple, open-source containerized applications:

- [simple-go-api](#), written in the [Go programming language](#)
 - Container: registry.gitlab.com/akosma/simple-go-api:latest
- [catalog-dotnet](#), written in [C#](#) with the [ASP.NET framework](#).
 - Container: registry.gitlab.com/akosma/catalog-dotnet:latest
- [image-go-api](#), written in the [Go programming language](#), using the [Gin Web Framework](#) and generating images via [ImageMagick](#).
 - Container: registry.gitlab.com/akosma/image-go-api:latest
- [simple-deno-api](#), written in [TypeScript](#) and running with [Oak](#) under the [Deno runtime](#)
 - Container: registry.gitlab.com/akosma/simple-deno-api:latest

1.5. Recommended LinkedIn Learning Courses

To get the basic knowledge, we recommend the following three courses on [LinkedIn Learning](#):

- "[Learning Docker](#)" by Carlos Nunez
- "[Learning Kubernetes](#)" by Kim Schlesinger
- "[Learning Git and GitHub](#)" by Ray Villalobos

Chapter 2. GitHub Repository

The GitHub repository for this course is available here:

github.com/LinkedInLearning/learning-openshift-4407066

2.1. Branches

This repository has branches for each of the videos in the course. You can use the branch pop up menu in github to switch to a specific branch and take a look at the course at that stage, or you can add `/tree/BRANCH_NAME` to the URL to go to the branch you want to access.

You can clone the repository in your computer and use the `git checkout` command to switch between branches.

This is the complete list of branches:

- Chapter 1: Getting Started
 - [01_07: Video 7: Solution: show how to install CRC in Linux](#)
- Chapter 2: Standard DevOps Practices
 - [02_01: Video 1: Understanding OpenShift security](#)
 - [02_02: Video 2: Deploying and debugging containers](#)
 - [02_03: Video 3: Building containers from source code](#)
 - [02_06: Video 6: Solution: setup a CI/CD pipeline in your cluster](#)
- Chapter 3: Advanced Cloud Native Apps
 - [03_02: Video 2: Serverless applications with Knative](#)
 - [03_03: Video 3: Microservices and Service Mesh](#)
 - [03_04: Video 4: Challenge: Deploy a Microservices application](#)
 - [03_05: Video 5: Solution: Deploy a Microservices application](#)
- Chapter 4: Scaling and Monitoring Apps
 - [04_01: Video 1: Logging](#)
 - [04_02: Video 2: Manual, Horizontal, and Vertical Scaling](#)
 - [04_03: Video 3: Monitoring apps with Prometheus](#)
 - [04_05: Video 5: Solution: Monitor an application](#)

2.2. Beware!

When switching from one exercise files branch to the next after making changes to the files, you may get a message like this:

```
error: Your local changes to the following files would be overwritten by checkout:
[files]
```

Please commit your changes or stash them before you switch branches.

Aborting

To resolve this issue, add changes to git using `git add` and commit them using `git commit`.

Getting Started

This section provides a brief overview of the main features of OpenShift, and shows how to get access to OpenShift clusters during your education.

Chapter 3. What is OpenShift?

During the past decade, businesses of all sizes have adopted [Kubernetes](#) as the preferred platform for running their applications. However, running Kubernetes presents quite a few challenges to developers and operation teams.

These challenges range from availability to configuration to security and monitoring. In many ways, the standard open-source Kubernetes project lacks tooling and support for enterprise use.

Here are some examples of specific Enterprise needs not covered by standard Kubernetes clusters:

- Enterprises require user and group management for various reasons, ranging from regulatory to organizational.
- Many team members prefer visual management tools, standard in other enterprise products, instead of text-based terminals.
- Security engineers must ensure that individual containers running on corporate clusters are incapable of privilege escalation.
- DevOps teams require many development and deployment tools to work efficiently with containers, such as CI/CD pipelines and container registries. Budgeting and provisioning such tooling from different vendors is complicated and expensive.
- Operation teams require standardized telemetry from their clusters.

A whole market of Kubernetes-based tooling blossomed in the past decade, supporting businesses with products covering those needs, but only some of those vendors provide complete enterprise-ready solutions. Red Hat is among those few.

3.1. What is Red Hat OpenShift?

Red Hat OpenShift is an enterprise-class platform built upon Kubernetes and provides a full DevOps product ready to use. It is designed with high availability and security in mind and integrates a whole host of DevOps tools in a single package.

Among its features, we can find:

- Built-in user and group management.
- Tighter security requirements for containers.
- More robust namespace isolation through projects.
- An integrated visual management console.
- An embedded container registry.
- A CI/CD pipeline system.
- A built-in Cloud Native application store featuring ready-to-use applications bundled as Kubernetes Operators.
- Integrated management and logging features.

Red Hat OpenShift clusters feature more robust security defaults than stock Kubernetes. They can also run in high-availability mode, ensuring the best possible performance for your applications. These characteristics set OpenShift apart as an excellent Kubernetes platform for enterprise users.

The latest version of OpenShift available at the time of this writing is 4.12.

3.2. Is Red Hat OpenShift Open Source?

Red Hat OpenShift is a commercial product based on an open-source project called [OKD](#). This acronym means "OpenShift Kubernetes Distribution" and is publicly available for everyone to inspect and contribute. Like the upstream Kubernetes project, OKD developers use the Go programming language.

3.3. How can I run OpenShift?

Today, Red Hat OpenShift is available through various mechanisms and formats:

- DevOps teams can install it in their data centers "on-premise."
- Major hyperscalers such as AWS, Azure, Google Cloud Platform, and IBM Cloud offer managed Red Hat OpenShift installations.
- Developers can either run OpenShift locally on their workstations using Red Hat OpenShift Local, also known as CRC or "Code-Ready Containers"
- They can also request a 30-day trial OpenShift cluster, offered by Red Hat, at no charge, for testing and evaluation purposes.

Red Hat OpenShift is an integrated Platform-as-a-Service for enterprise users based on Kubernetes. It is tightly integrated with advanced security settings, developer tooling, and monitoring mechanisms, allowing DevOps teams to be more productive.

Chapter 4. OpenShift-only Custom Resource Definitions

Red Hat OpenShift is a complete DevOps platform extending Kubernetes in various ways. It bundles a constellation of Custom Resource Definitions (CRDs) to make the life of developers and cluster administrators easier.

Let us talk first about the CRDs only available on OpenShift.

4.1. Project

An OpenShift Project is similar to a Kubernetes namespace, but more tightly integrated into the security system of OpenShift through additional annotations.

```
apiVersion: project.openshift.io/v1
kind: Project
metadata:
  name: linkedin-learning-project
  annotations:
    openshift.io/description: "Project description"
    openshift.io/display-name: "Display name"
```

4.2. Route

The OpenShift Route object was one of the primary inspirations during the development of the **Ingress** object. In OpenShift, **Ingress** and **Route** objects work together to ensure your applications are available outside the cluster.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: my-route
spec:
  host: example.com
  port:
    targetPort: 8080
  to:
    kind: Service
    name: my-service
```

4.3. DeploymentConfig

Kubernetes' Deployment object draws its inspiration from the OpenShift-only DeploymentConfig object. Both are based on a ReplicationController object, supporting the deployment and

configuration of an application. But DeploymentConfig objects can also specify deployment strategies and trigger changes or application upgrades from the command line.

```
apiVersion: v1
kind: DeploymentConfig
metadata:
  name: mydc
spec:
  replicas: 1
  template:
    spec:
      containers:
        - name: container
          image: image-name:version
```



Red Hat does not recommend creating DeploymentConfig objects, and to create standard Deployment objects instead. But they are common enough in older OpenShift installations, which is why they deserve to be mentioned.

4.4. BuildConfig

OpenShift provides a CRD called BuildConfig that describes the definition and configuration required for the build process of an application.

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: my-build
spec:
  triggers:
    - type: "GitHub"
  source:
    git:
      uri: "https://github.com/user/project"
```

The typical outputs for such build processes are container images conveniently stored in the integrated OpenShift container repository for developers to inspect and debug.

Chapter 5. OpenShift-only Command-Line Tools

Let us talk now about the two OpenShift command-line tools.

5.1. oc

OpenShift includes a feature-rich graphical user interface, but most DevOps engineers prefer the speed and convenience of the `kubectl` tool to perform actions on their clusters.

OpenShift bundles a tool called `oc` with many of the same options as `kubectl` but with a more profound knowledge of OpenShift-only objects and features.

Like `kubectl`, the `oc` tool can use Kustomize templates. Still, it offers an additional "oc process" feature, which can read values from the local environment: export some environment variables on your terminal, call "oc process," and your YAML manifests will include their values.

5.2. odo

Another OpenShift-only command-line tool is the `odo` or "OpenShift Do" application. `odo` helps software developers in the process of creating containerized applications running on OpenShift, abstracting Kubernetes and OpenShift concepts. `odo` can quickly create and deploy containers including applications written in various programming languages and frameworks, from Java to Go to .NET to Python and JavaScript.

DevOps engineers can download the `oc` and `odo` tools directly from the OpenShift cluster through a menu on the web console.

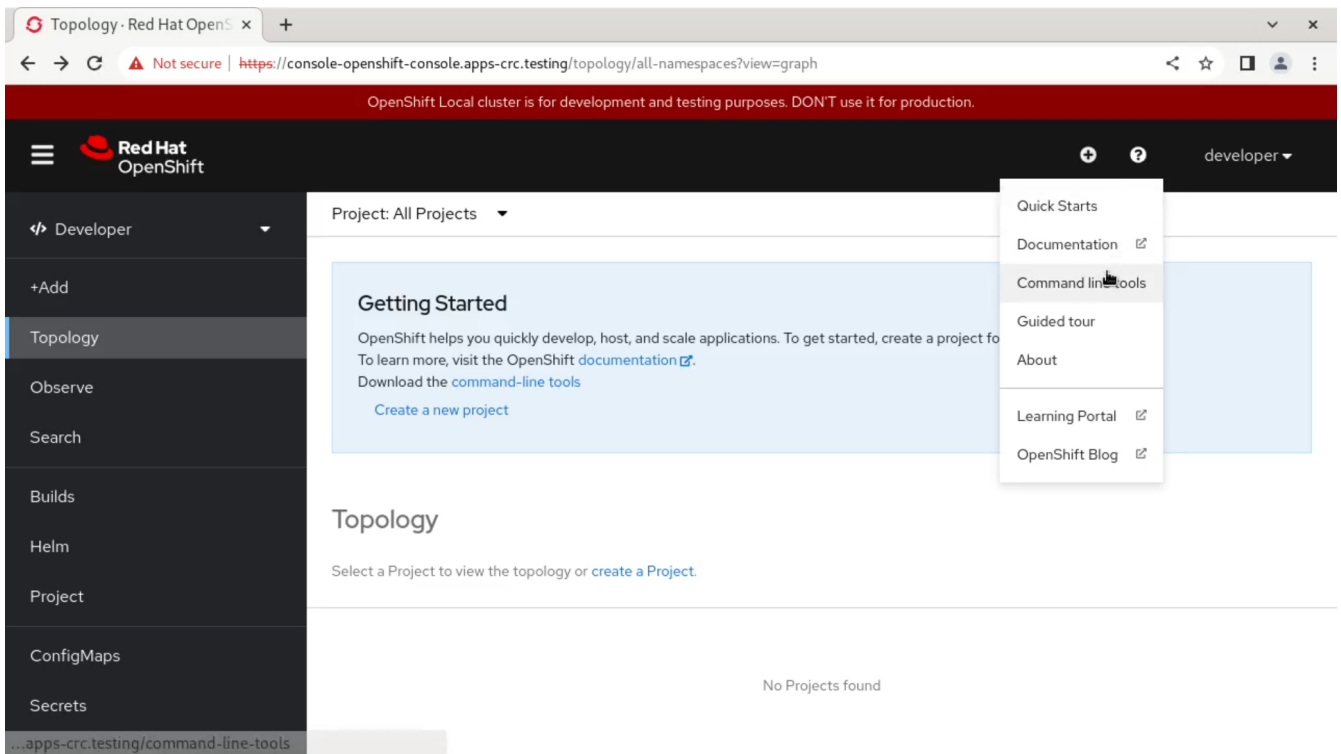


Figure 1. Download the OpenShift CLI tools directly from the Web Console

Chapter 6. Red Hat OpenShift Developer Sandbox

There is no better way to learn Red Hat OpenShift than by experiencing it firsthand.

One of the simplest ways to do this is to use the Developer Sandbox for Red Hat OpenShift, a free service by Red Hat with which developers can get access to a fully managed cluster for 30 days. The Developer Sandbox for Red Hat OpenShift is an ideal solution for those interested in knowing more about OpenShift with the least effort.

With the Developer Sandbox for Red Hat OpenShift, developers can deploy their applications in a secure and fully managed environment. Red Hat even provides guided activities, but developers can deploy their container images and use their deployment manifests.

You only need a free Red Hat developer account to use the Developer Sandbox for Red Hat OpenShift. Your free Red Hat developer account will give you access to many resources and software, freely available at developers.redhat.com.

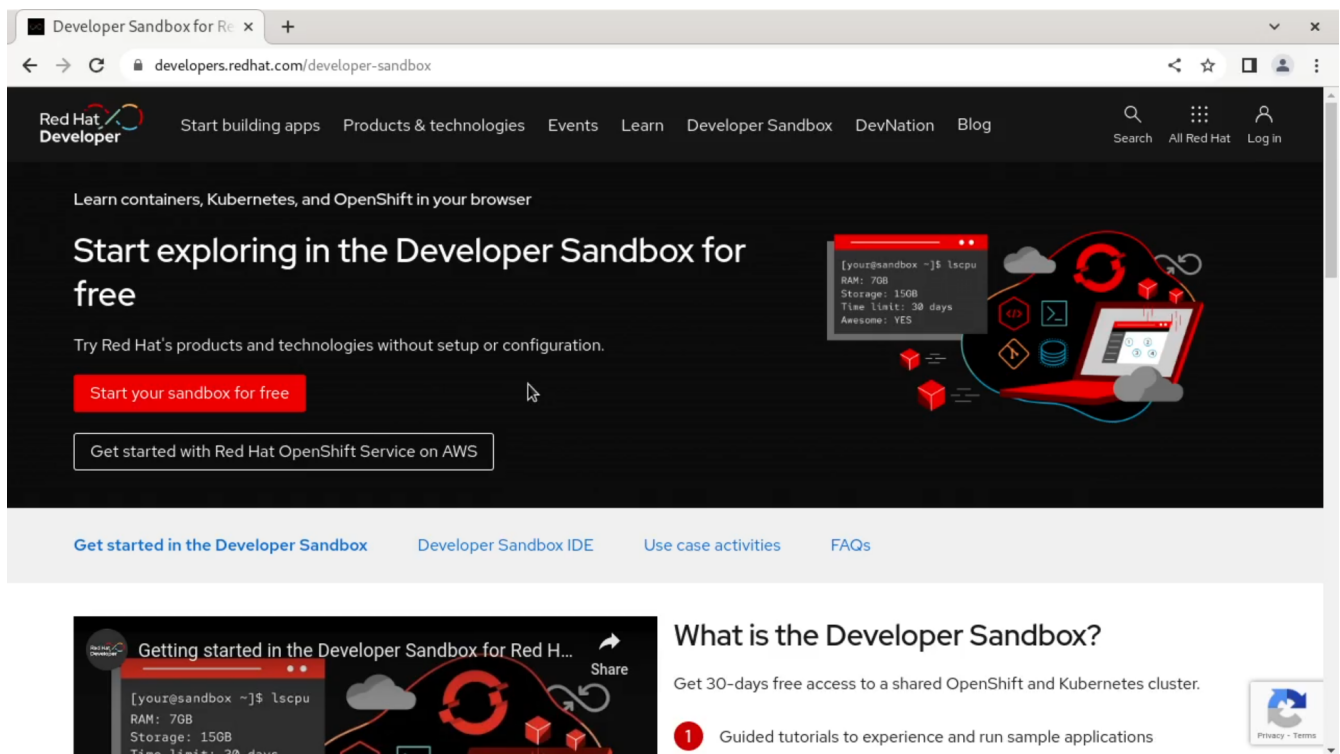


Figure 2. Red Hat Developer Sandbox home page

6.1. Demo

Go to developers.redhat.com/developer-sandbox and click on the **[Start your sandbox for free]** button. Log into your account, and if this is the first time you have accessed the Developer Sandbox, Red Hat will ask you to confirm your account. Confirmation is required to avoid service abuse. The easiest way to verify your account is by entering a code sent via SMS. After entering the code, you'll receive a confirmation email. Depending on the system's load, this verification can take some time.

Once confirmed, you can navigate with your browser to developers.redhat.com/developer-sandbox

and log in directly to your very own temporary OpenShift cluster! It's that simple.

6.2. Dev Spaces

An exciting feature of the Developer Sandbox for Red Hat OpenShift is Red Hat OpenShift Dev Spaces, a fully integrated developer environment within Red Hat OpenShift, allowing developers to create and deploy applications in various programming languages and frameworks from within the OpenShift cluster.

Red Hat OpenShift Dev Spaces also includes the web version of Microsoft Visual Studio Code, the popular text editor for developers. Visual Studio Code allows developers to work in a professional environment, including Git support, deploying their applications in a few clicks on a ready-to-use OpenShift cluster.

You can create Red Hat OpenShift Dev Spaces workspaces in various technologies: Go, .NET, Java (using the Quarkus framework), Node.js, Python, (©), C++, Rust, PHP, and Scala, among many others.

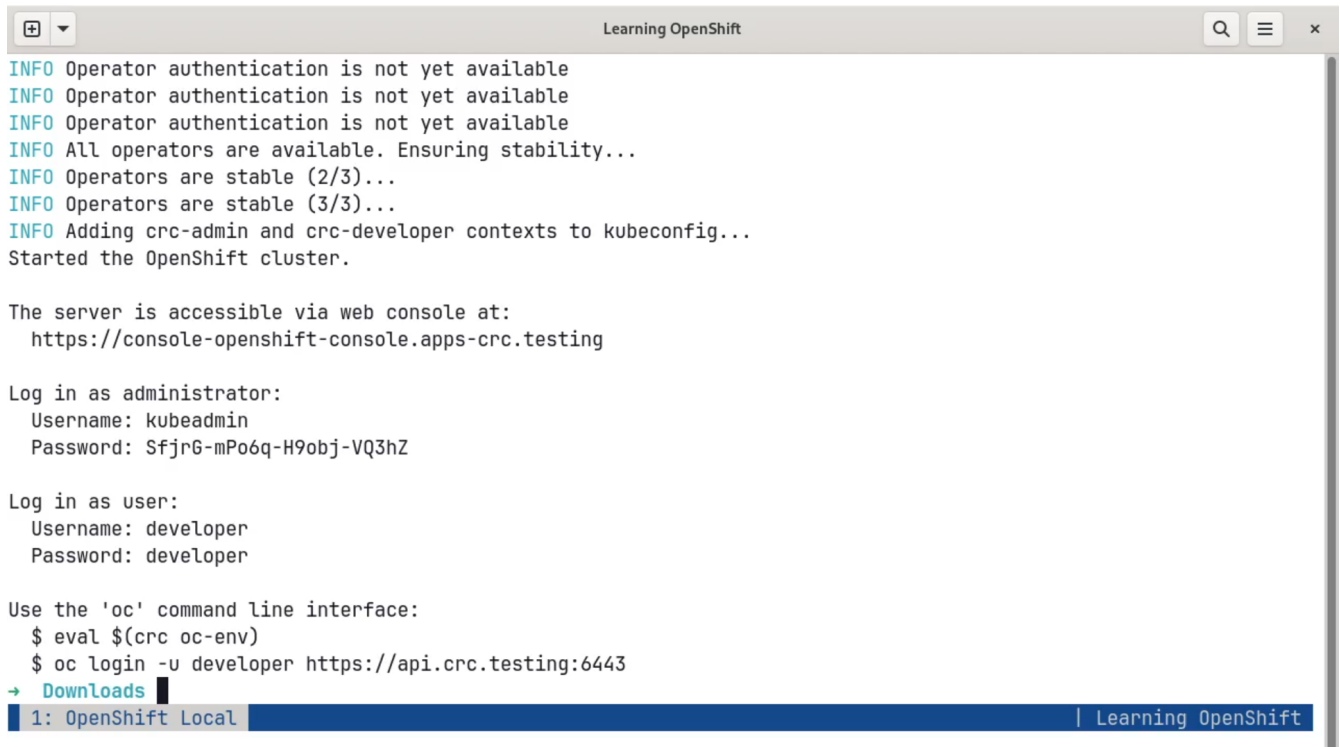
Dev Spaces also integrates with the `odo` command line tool through an auto-generated `Devfile`, specifying all the dependencies and steps required to build and deploy your application in whichever programming language you have chosen.

You can learn more about the `Devfile` specification at devfile.io.

Chapter 7. Red Hat OpenShift Local

Investigating OpenShift is much simpler when having access to a cluster. Red Hat OpenShift Local, previously known as CodeReady Containers or CRC, is a self-contained application available for computers running Windows, Mac, and Linux. Developers can download and run OpenShift Local and access various features in a minimalistic environment geared for developers and testers.

OpenShift Local is not prepared for production use, instead using the current workstation as both Control Plane and Worker Node. Among various optimizations, OpenShift Local runs by default with disabled monitoring features, which can be enabled if needed.



```
Learning OpenShift
INFO Operator authentication is not yet available
INFO Operator authentication is not yet available
INFO Operator authentication is not yet available
INFO All operators are available. Ensuring stability...
INFO Operators are stable (2/3)...
INFO Operators are stable (3/3)...
INFO Adding crc-admin and crc-developer contexts to kubeconfig...
Started the OpenShift cluster.

The server is accessible via web console at:
  https://console-openshift-console.apps-crc.testing

Log in as administrator:
  Username: kubeadmin
  Password: SfjrG-mPo6q-H9obj-VQ3hZ

Log in as user:
  Username: developer
  Password: developer

Use the 'oc' command line interface:
  $ eval $(crc oc-env)
  $ oc login -u developer https://api.crc.testing:6443
→ Downloads
1: OpenShift Local | Learning OpenShift
```

Figure 3. Terminal at the end of the CRC startup process

7.1. Requirements

OpenShift Local requires the following minimum operating system configurations:

- Windows 10 Fall Creators Update (version 1709) or later.
- macOS 11 Big Sur or later.
- Red Hat Enterprise Linux (RHEL), Fedora, or CentOS 8 or later.

These requirements are valid as of the publication of this video, and may change over time. We recommend you to check the official Red Hat OpenShift Local documentation for an updated list of requirements at [the official documentation website](#).



Regarding Linux, even if Red Hat does not officially support them, OpenShift Local can run on other distributions, such as Ubuntu or Debian, with minor caveats. Running OpenShift Local on any Linux distribution requires a few additional software packages to be installed through your default package manager. The

documentation at crc.dev/crc has more information about this subject.

7.2. Hardware Requirements

In terms of hardware, OpenShift Local has some strict requirements. Your system must use a recent Intel CPU (except for Macs, where Apple Silicon machines are supported) with at least four physical cores and have at least 16 GB of RAM. Be aware that the base installation of OpenShift Local requires at least 9 GB free to start. Of course, to run other applications on OpenShift Local, you will need more RAM, so using a computer with at least 32 GB of RAM is strongly recommended. OpenShift Local also requires at least 35 GB of free disk space for its installation. The memory requirements are likely to increase in the future, so please check the documentation at crc.dev for more up-to-date information.

7.3. Installation

To install OpenShift Local, open your web browser and navigate to console.redhat.com/openshift/create/local. Download the latest release of OpenShift Local and the "pull secret" file. The latter is a file containing a key identifying your copy of OpenShift Local to your Red Hat Developer account.

Unzip the file containing the OpenShift Local executable, and using your terminal, run the command `crc setup`. This command will prepare your copy of OpenShift Local, verifying requirements and setting the required configuration values.

Once the `crc setup` command is ready, launch `crc start`. Running `crc start` can take a long time, around 20 minutes, on a recent PC.

Once started, access the OpenShift Web Console with the `crc console` command, which will open your default browser. OpenShift Local uses the `developer` username and password to log in as a low-privilege user, while the `kubeadmin` user uses a random-generated password. Use the `crc console --credentials` command to find the credentials required to log in as the `kubeadmin` user.

OpenShift Local allows developers to perform various everyday tasks as if it were a standard OpenShift cluster, like deploying applications and configuring and triggering CI/CD pipelines.

OpenShift Local can take a long time to start. For that reason, you can run `crc stop` to freeze the cluster in its current state to resume operations later. On the other hand, the command `crc delete` completely removes the OpenShift Local cluster from the workstation.

Red Hat OpenShift Local uses two DNS domain names: `crc.testing` and `apps-crc.testing`. The first provides functionalities required internally by the cluster, while the second exposes OpenShift applications deployed on the cluster to the outside world.



Ubuntu Linux users should modify their `/etc/hosts` file so that both domain names point to the local IP; this is something RHEL, Fedora, and CentOS users don't need to do.

To enable monitoring on your OpenShift Local instance, you should execute the command `crc config set enable-cluster-monitoring true` before running `crc setup`.



There is a demo script for the installation of CRC on Linux in the [01_07 branch](#) of the [GitHub repository](#) for this course.

Standard DevOps Practices

In this section you'll review how to perform some common DevOps practices with Red Hat OpenShift.

Chapter 8. Understanding OpenShift Security

OpenShift is, as per its definition, a container platform, and its role is to run and orchestrate lots and lots of containers. But containers are nothing more than isolated Linux processes, running under very tight conditions and the control of several agents.

The biggest threat on such a platform is that of privilege escalation. Rogue containers might include malicious code that tries to break out from the container where it is running, using mechanisms such as the "mknod" or "setuid" system calls. Privilege escalation is a real threat, and there have been many documented cases of Kubernetes clusters suffering from such attacks.

Red Hat has designed OpenShift to be secure by default by placing more restrictive conditions around containers than other Kubernetes distributions. Such security defaults have a common side effect, well-known by developers working on OpenShift: many container images that work with other Kubernetes distributions do not work with OpenShift. Such behavior can be confusing for developers, and in this section, we will review some of those mechanisms so that your container images can run on OpenShift without issues.

8.1. Users and Roles

To be able to operate on an OpenShift project, a DevOps engineer should have at least the "edit" role.

There are also service accounts, used in conjunction with external services such as CI/CD systems, which must be able to access the cluster in a limited but secure way.

8.2. Security Context Constraints

OpenShift is configured by default using Security Context Constraints, or SCCs. The values of the SCCs provided by OpenShift are secure by default. Still, they are simply Custom Resource Definitions (CRDs), and your cluster administrator can technically modify them, even if this practice is far from recommendable.

Always check with your cluster administrator to ensure the default security settings of your cluster before building or deploying your cluster images.

8.3. Base Images

All container images "inherit" lots of their behavior from their base images. As such, DevOps developers must always consider their images' origin and safety. If your container image is not working on OpenShift, some setting on your base image prevents you from reaching your goal. Review your base images carefully, and ensure they are not running privileged processes or requesting resources unavailable to regular users.

Red Hat provides a range of base images called the Red Hat Universal Base Images, or UBI. These

images, distributed through the Red Hat Quay registry, are validated by Red Hat and can be considered a safe option for your container images.



There are `Dockerfile` examples of non-OpenShift compatible containers in the `02_01` branch of the [GitHub repository](#) for this course.

8.4. Building Images

Whenever you build images for OpenShift, remember to use the `USER` command with a value greater than 1000; this will instruct the container runtime to run your application not as `root` but as a regular user. Your containers must run using non-root accounts to be usable within OpenShift.

Suppose you do not specify a default `USER` for your image. In that case, OpenShift will run your container using arbitrarily assigned user IDs, which can be problematic if your image requires storing data on a file system.

8.5. Network Ports

Another common problem with container images on OpenShift is their use of system network ports, ranging from 0 to 1023. Those ports are privileged ports, and only processes running under the `root` account can use them. Because your images cannot run on OpenShift as `root`, you cannot use those ports.

Remember to end your Dockerfiles with the instruction `EXPOSE` and a number greater or equal to 1024.

A Dockerfile to create containers compatible with Red Hat OpenShift

```
FROM docker.io/library/nginx:stable

EXPOSE 8080

RUN \
  # Support running as arbitrary user
  chmod g+rx /var/cache/nginx /var/run /var/log/nginx && \
  # Users are not allowed to listen to privileged ports
  sed -i.bak 's/listen\(.*\)80;/listen 8080;/' /etc/nginx/conf.d/default.conf && \
  # Comment user directive as master process is run as different user
  sed -i.bak 's/^user/#user/' /etc/nginx/nginx.conf

USER 1001:0
```

8.6. Storage

If your container must access data stored on a file system, you might need to `chgrp` and `chmod` the paths in your persistent volumes so that your container can access them. Pay attention to the user ID of the container; run your containers with a defined ID if possible.

8.7. Apply "12 Factor" Best Practices

Remember to apply the principles of the Twelve-Factor App when creating your containers. In particular:

1. Remember to store configuration values as environment variables.
2. Separate build and run stages for your images using multiple-step builds.
3. Deploy a single binary per container.
4. Log everything to "stdout."
5. Divide your application into various stateless processes.

And always keep your container images as small as possible. Remember that OpenShift can scale your deployments vertically if needed, just like any Kubernetes cluster, so using small and lean images will make your life easier.

Learn more about the Twelve-Factor App at 12factor.net.

Chapter 9. Deploying and Debugging Containers



The source code for this section is contained in the [02_02 branch of the GitHub repository](#) for this course.

The whole point of OpenShift is to be able to deploy, run, and monitor containerized applications. DevOps engineers can deploy containers in OpenShift clusters through various means, for example:

- Using a YAML manifest.
- Deploying a single container using the web console.
- Building a project stored in a Git repository anywhere on the Internet and deploying the resulting container.
- Using the integrated CI/CD pipelines.
- Using the `odo` tool together with "Devfiles."

Each approach has pros and cons; in this chapter, we will review how to use the web console and how to use the `odo` tool.

9.1. Deploying Applications with the Web Console

DevOps engineers can deploy applications immediately using the Web Console. Launch your CRC instance and open the web console in your preferred web browser. The URL of the OpenShift web console is "https://console-openshift-console.apps-crc.testing."

The OpenShift Web Console offers two major perspectives:

- The "Administrator" perspective.
- And the "Developer" perspective.

For this explanation, select the "Developer" perspective.

The first time you open the Developer perspective, a popup invites you to follow a user interface tour.

On the left-hand side, the perspective menu shows an entry titled "Add," which, as the name implies, provides various mechanisms to deploy applications on an OpenShift cluster.

The "Add" screen shows the various ways DevOps engineers can deploy applications on a cluster:

- Using the Developer Catalog, browsing and choosing among a long list of available databases, message queues, and other valuable components to build applications, or entering your preferred Helm chart repository to extend the catalog.
- Specifying the URL to a specific container on any standard container registry.
- Specifying the URL of a source code project stored on a Git repository, for example, but not

limited to GitHub, GitLab, Gitea, or other locations.

- Importing YAML directly or even a JAR file with a Java application.

Let us select the "Container Image" option, where we can specify the URL of a ready-to-use container.

Enter the URL of the container on the field, and click on the [**Create**] button at the bottom of the page. You do not need to change any other value on the form.

A few seconds later, depending on the size of the container and the speed of your Internet connection, OpenShift will have pulled the container and deployed it onto your cluster. This deployment will include the usual standard elements: a "Deployment" object, a "Service" object, and a "Route."

OpenShift offers a visual representation of the applications running on your project: click on the icon of your container, and you will see a panel opening on the right side of the screen. This panel will include the URL automatically assigned to your deployment, and clicking it will show the application in action in another browser tab.

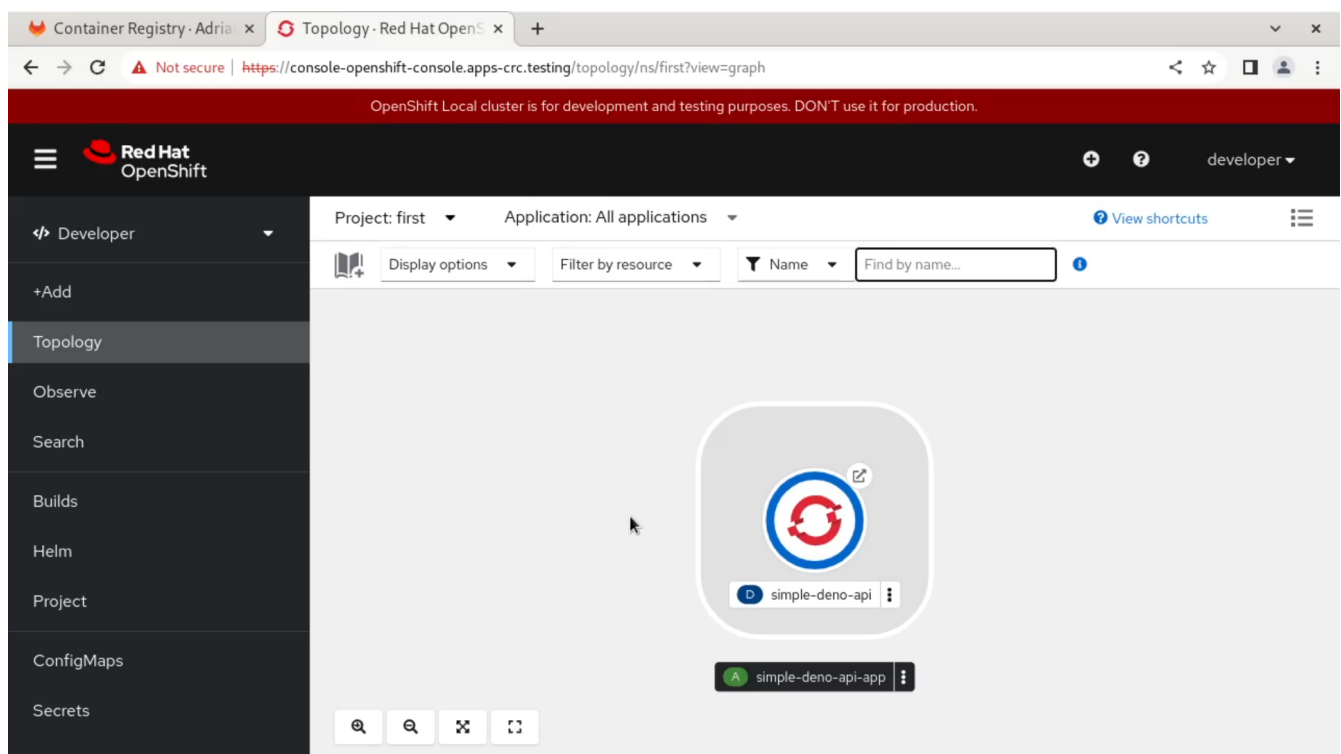


Figure 4. Topology screen on Red Hat OpenShift

9.2. Creating and Debugging Applications with the **odo** Tool

With the **oc** tool, Red Hat provides another one geared toward software developers: the **odo** tool. Developers can use the **odo** tool to create applications using "Devfiles," particular files named "devfile.yaml" based on an open standard available at the Devfiles website. Devfiles contain information about your application's programming language, dependencies, and other essential details.

The `odo` tool is not available by default on your command line, but you can download it from the "Help" menu on the OpenShift Web Console through the "Command line tools" entry. Click on the "Download odo" link at the bottom, and select the version of odo that corresponds to your system.

The "`odo catalog list components was`" command shows the various programming languages and frameworks supported off-the-box by "odo."

The `odo init` command prompts the user for a new application using many programming languages: .NET, Go, Java, JavaScript, PHP, Python, and TypeScript. The last command generates a scaffold ready to be populated with the required logic. Finally, the `odo push` command builds and pushes the container to the OpenShift container registry, deploying a copy.

Developers can debug applications built with Node.js or Java directly from the command line using the `odo` tool. The command `odo debug port-forward` opens the required ports for developers to connect a debugger to their container, usually with the help of an IDE, and debug their application with breakpoints and step-by-step execution.

Chapter 10. Building Containers from Source Code



The source code for this section is contained in the [02_03 branch of the GitHub repository](#) for this course.

Software developers have adopted Git as the de facto source code management system in the past 15 years. The good news is that OpenShift knows Git well, so you can deploy applications directly from your source code repositories without much effort.

10.1. Deploying

It is straightforward to deploy an application from your source code repository. First, open the OpenShift web console and log in as the "kubeadmin" user. Then navigate to the Developer perspective. Then create a new project to encapsulate and easily manage your work. Call this project "sourcecode."

You should now see the "+Add" menu entry on the left side of the screen. One of the options in that screen is the "Import from Git" entry. Click on it, and paste the URL of a project, for example, gitlab.com/akosma/simple-deno-api.git.

As soon as you paste the URL, OpenShift will immediately analyze the structure and programming language of the project and automatically recommend options for its build process. In our case, it's a small application built with the Go programming language, and as such, it will advise the options shown on the screen.

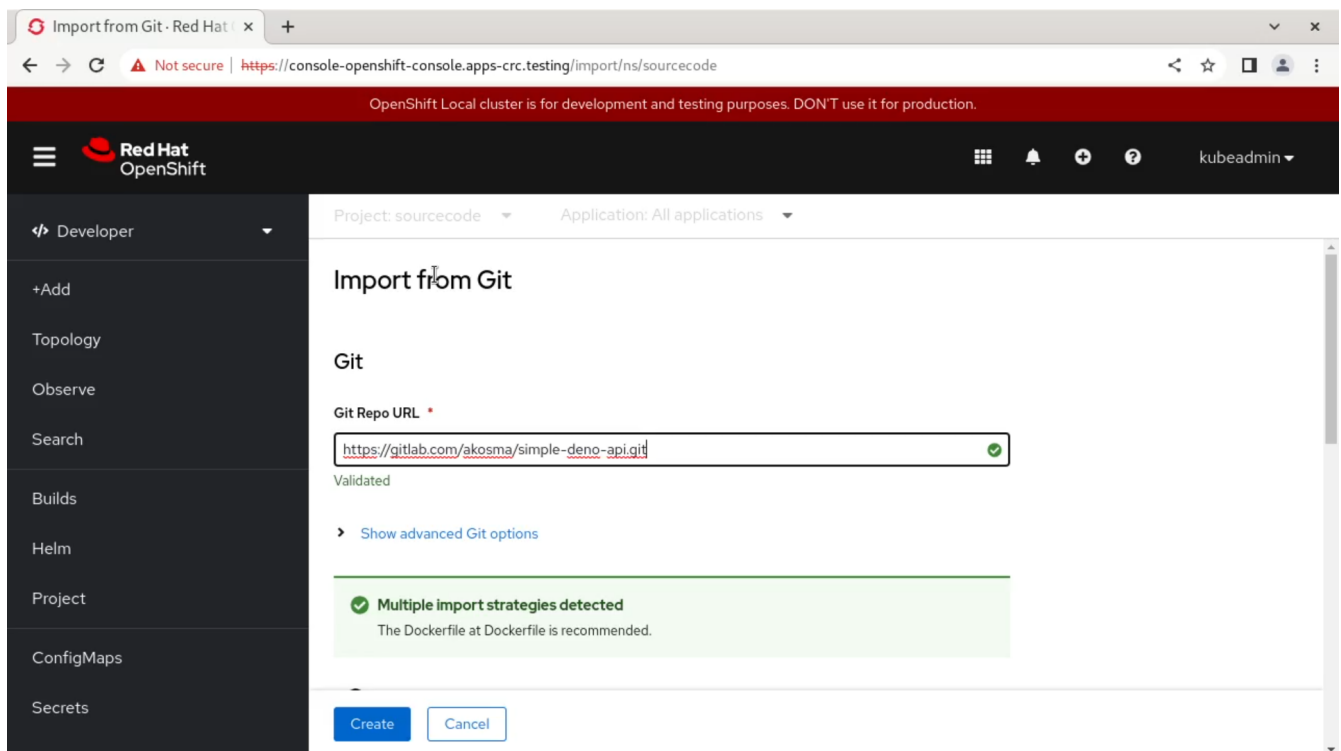


Figure 5. Deploying a project directly from its Git repository

This particular example doesn't require more configurations than the ones shown on the screen; click the **[Create]** button.

After a few seconds, you will see your application running on the "Topology" screen. OpenShift will download the source code and trigger your project's build. Click on the Topology screen icon to see the "Build" section, indicating that a build is running. The compilation and deployment of your application can take some time, depending on the complexity of the source code and the programming language used.

Once the build has finished, on the same pane, you will see a route available under the "Routes" section. Click on it, and you will see your application in action.

10.2. Container Registry

OpenShift has built your application source code, and the product of this build process is a container. You can see the container that OpenShift made for you on the "Administrator" perspective, selecting the "Builds" menu and then the "ImageStreams" menu entry.

OpenShift includes a container registry; developers can use it as any other registry from outside the cluster. Let us use "podman" to access the container registry and run the container locally on your workstation.

Users must have the "registry-editor" and the "system:image-builder" roles to access the container registry. Since we're connected to the Web Console using the "kubeadmin" user, we can provide those roles directly from the user interface without using the command line.

Navigate to the "User Management" section and select "RoleBindings." Click on the **[Create binding]** button, and fill the form using the following values:

- Name: `developer-sourcecode-registry-editor`
- Namespace: `sourcecode`
- Role name: `registry-editor`
- Subject: `User`
- Subject name: `developer`

Do the same for the "system:image-builder" role, using a different "Name" field on the top of the form.

Now on the terminal, log in as "developer" and use the following commands:

```
PROJECT=sourcecode
REGISTRY="default-route-openshift-image-registry.apps-crc.testing/${PROJECT}"
podman login --tls-verify=false -u unused -p $(oc whoami -t) ${REGISTRY}
podman pull ${REGISTRY}/hello-git:latest
podman run --rm --publish 5000:8080 ${REGISTRY}/hello-git:latest
```

When logging into the container registry provided by CRC, we must specify the "--tls-verify=false"

option to avoid errors caused by self-signed certificates. To prevent passing this entry every time we use the command, we can add the CRC registry to the configuration of our system as a trusted registry. On RHEL, you can do that by editing the file `/etc/containers/registries.conf` or your local equivalent, `~/config/containers/registries.conf` and adding the following snippet:

```
[[registry]]
location="default-route-openshift-image-registry.apps-crc.testing"
insecure=true
```

10.3. Updating the Application

The purpose of source code is to evolve continuously. Clone your Git project using your preferred method, and perform a minor modification in the source code, changing the message displayed on the screen. Then commit and push your changes to your repository.

On the OpenShift Web Console, return to the "Developer" perspective and select the "Topology" screen. You will see your application running. Click on the icon, and you will see, in the "Builds" section, a button labeled **[Start Build]**. Click on it and wait. This second build will re-fetch the code from the repository and will redeploy the application automatically for you.

Click on the application route to see your application displaying a new message. But it would be great if we didn't manually trigger a build.

Chapter 11. Using CI/CD Pipelines

CI/CD is "Continuous Integration and Continuous Delivery," an integral element of the DevOps world. Many projects provide CI/CD features in the Cloud Native world today, such as Flux, Argo CD, Jenkins, and Tekton.

OpenShift includes the Tekton open-source project of the Continuous Delivery Foundation to provide a complete CI/CD environment ready to use.

11.1. Install Tekton

To use CI/CD pipelines in OpenShift, your administrator must install the "Red Hat OpenShift Pipelines" operator and make it available to all namespaces. The operator appears in the "Operators" menu entry, and the kubeadmin user can install it on CRC using the default options by clicking on the **[Install]** button with default values. The installation of this operator can take a few minutes to complete, depending on the speed of your machine.

When the operator is ready, a "Pipelines" item appears in both the Developer and the Administrator perspectives of the OpenShift web console.

Tekton provides its command-line tool called "tkn." OpenShift users can download tkn directly from the "Command Line Tools" screen of the help menu.

Tekton is a complete CI/CD system, including the following concepts:

- Tasks are individual operations, such as cloning source code repositories or building container images.
- Pipelines are sets of tasks executed in sequence or parallel.
- Workspaces are abstracted storage elements where tasks in a pipeline can share state.

Developers can create Tekton objects such as Tasks, Pipelines, and Workspaces using either YAML or the visual editor provided by the OpenShift web console.

11.2. Example

Once your administrator has enabled the Red Hat OpenShift Pipelines operator on your cluster, the developer user can use the same "Import from Git" option we used in the previous section.

This time, after entering the URL of the Git project you would like to deploy in the cluster, you will see a new section called "Pipelines," with a checkbox labeled "Add pipeline."

Check the checkbox and click the "Hide pipeline visualization" link to display the suggested pipeline structure.

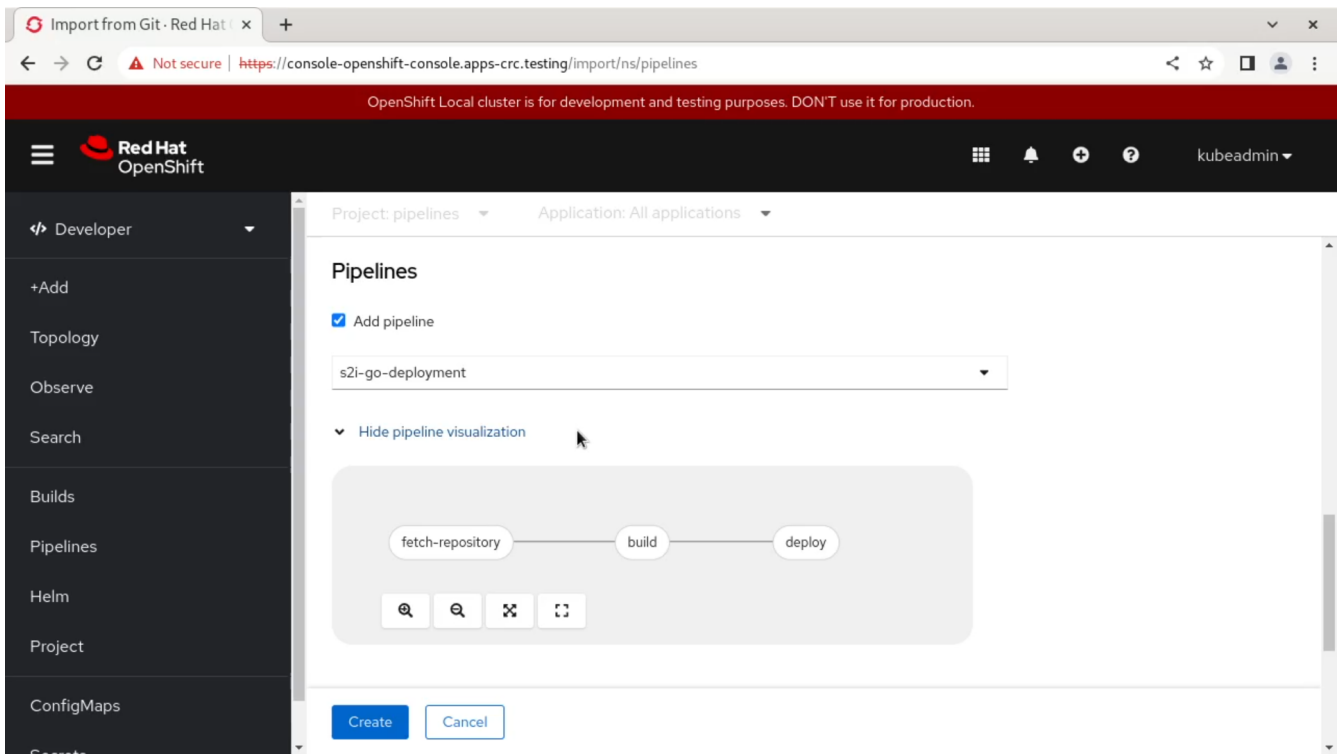


Figure 6. Visualization of a Tekton pipeline

The pipeline will first fetch the application's source code; then, it will build it, generating a container image to be stored locally on your project. And finally, it will deploy it.

Leave all other options intact and click the **[Create]** button at the bottom of the screen.

After a few minutes, the "Topology" screen will display two icons; the first is your application, built and running; the second is an "Event Listener Sink" or "Pipeline Trigger," offering a publicly available URL. You can use this URL in the configuration of your project (for example, in GitHub or GitLab) to trigger the rebuild and redeployment of the project as soon as your code changes.

Tekton offers `EventListener` and `Trigger` objects that can be called from outside the cluster, for example, from GitHub or GitLab, so that each time your source code changes, a new `PipelineRun` starts, and you could even request a redeployment of your code in production.

However, if you use CRC, the trigger URL is a subdomain of the "apps-crc.testing" domain, only accessible from your local machine. There are ways to expose this URL using a reverse proxy such as ngrok, but such a task is outside of the scope of this course.

Also of interest, Tekton offers a Visual Studio Code extension to help developers create the YAML representations of Workspaces, Tasks, Pipelines, and other objects.

11.3. Inspecting the Pipeline

Click on the "Pipelines" entry on the left-hand side menu of the OpenShift web console. This screen displays the list of Tekton pipelines defined in the cluster at any time. It also conveniently shows the status of the pipeline; a full green bar means, of course, that the last execution of the pipeline was a success.

Select the item on the list with the same name as the application created previously. The "Pipeline

details" screen shows the pipeline structure, which is precisely the same as seen once during the deployment creation.

The tab "PipelineRuns" contains the history of all the times when the pipeline was executed, including its status and result. Select the last item on that list and click on any pipeline steps. A new pane will show the logs of the execution of the pipeline, including all the operational records.



The branch [02_06](#) of the [GitHub repository](#) for this course contains an example of a CI/CD pipeline built with YAML files.

Advanced Cloud Native Apps

In this section you will learn some advanced topics, like templates, operators, serverless with Knative, and service mesh with Istio.

Chapter 12. Templates and Operators

Let's talk about OpenShift 4.12 templates and operators, both powerful ways to create and deploy applications on OpenShift.

12.1. Templates

OpenShift templates are YAML or JSON files that describe the desired state of application components, such as pods, services, routes, and build configurations. You can use templates to define reusable and parameterized application configurations that can be instantiated with a single command or through the web console.

Templates can help you simplify and automate the creation of complex applications, such as microservices, that consist of multiple components that need to communicate with each other. You can also use templates to share best practices and common patterns for deploying applications on OpenShift.

```
apiVersion: template.openshift.io/v1
kind: Template ①
metadata:
  name: postgresql
parameters:
  - name: APPLICATION_NAME ②
    description: "The application name"
    value: postgresql
  - name: POSTGRES_USER
    description: "Username of the PostgreSQL user"
    value: user
objects:
  - apiVersion: v1
    kind: DeploymentConfig
    metadata:
      labels:
        app: ${APPLICATION_NAME} ③
        name: ${APPLICATION_NAME}
    spec:
      replicas: 1
      selector:
        app: ${APPLICATION_NAME}
      strategy:
```

- ① Definition of a template conforming to the OpenShift CRD.
- ② Declaration of an `APPLICATION_NAME` parameter.
- ③ Usage of the `APPLICATION_NAME` parameter.



You can see the full contents of a template in action in the [04_02 branch of the GitHub repository](#) for this course, in the file `00-postgresql-template.yaml`.

OpenShift 4.12 introduces some new features and enhancements for templates:

- Support for Helm charts, which are a popular way to package and distribute Kubernetes applications. You can now use Helm charts as templates on OpenShift and leverage the Helm CLI and the OpenShift web console to install and manage them.
- Improved template editing and validation in the web console, which makes it easier to create and modify templates with a graphical interface and provides instant feedback on syntax errors and parameter values.
- New template samples and quick starts, which provide ready-to-use examples of common application scenarios, such as deploying a Node.js application or a PostgreSQL database. You can access these samples and quick starts from the Developer Catalog or the Add page in the web console.

12.2. Operators

Operators are a powerful way to automate the deployment and management of Kubernetes-native applications on OpenShift. Operators are software extensions that act like a vendor's engineering team, monitoring the cluster state and making decisions in real time. Operators can provide automation at every level of the stack, from the platform components to the applications that run on top of it.

Operators are packaged, deployed, and managed through OperatorHub, a registry of certified operators from software vendors and open-source projects. Users can browse and install operators from the OperatorHub with a few clicks, and get updates and patches automatically. Operators can also expose configuration options through custom resource definitions (CRDs), allowing users to fine-tune their applications according to their needs.

OpenShift operators make it easier to run complex and stateful applications on Kubernetes, such as databases, message queues, or monitoring systems. Operators can handle tasks such as provisioning, scaling, backup, recovery, and upgrade, reducing the operational burden on users. Operators can also integrate with other OpenShift features, such as security, networking, and logging, to provide a consistent and seamless user experience.



To see operators in action, check the following chapters: [advanced/serverless.pdf](#), [monitoring/logging.pdf](#), and [devops/pipelines.pdf](#).

Chapter 13. Serverless with Knative



The source code for this section is contained in the [03_02 branch of the GitHub repository](#) for this course.

Serverless is a cloud computing model and architecture that divides the application logic into minimal units. These units have fast startup and shutdown times, and they scale automatically: according to demand, and the platform fully manages this scaling without human intervention.

Common examples of Serverless services are AWS Lambda and Azure Functions.

In the case of OpenShift, the Knative project enables Serverless functionality. You can learn more about Knative at their website: knative.dev.

Knative is an open-source project created by Google, and it has become a de facto standard for developing and deploying serverless services in Kubernetes.

Knative is encapsulated in an OpenShift operator called the "Red Hat OpenShift Serverless" operator. Your cluster administrator must install this operator.

Knative services are defined just like any other Kubernetes object using YAML. This example shows that Knative services exist in a separate custom resource definition from standard Kubernetes services.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: simple-deno-api
spec:
  template:
    metadata:
      name: simple-deno-api-v1 ①
      annotations:
        autoscaling.knative.dev/min-scale: "0"
        autoscaling.knative.dev/max-scale: "10" ②
        autoscaling.knative.dev/target: "3"
    spec:
      containers:
        - image: registry.gitlab.com/akosma/simple-deno-api:latest ③
```

- ① Knative services must also follow a versioning schema shown in the example, which allows various versions of the same service to be installed simultaneously on the same cluster.
- ② Developers configure Knative services autoscaling with annotations.
- ③ Knative uses standard containers as serverless units of work.

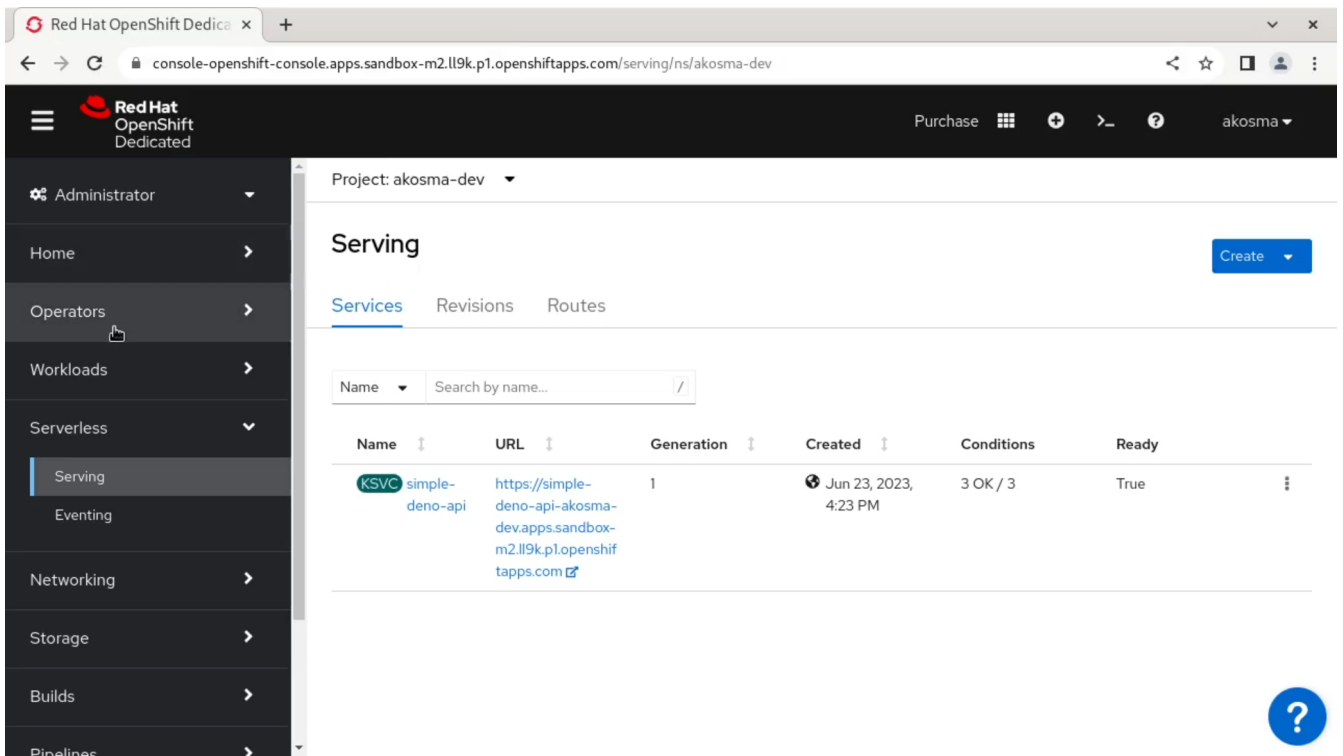


Figure 7. A Knative serverless service deployed on Red Hat OpenShift

Chapter 14. Microservices and Service Mesh



The source code for this section is contained in the [03_03 branch of the GitHub repository](#) for this course.

Microservices are a very common architectural pattern these days, in which complex applications are decomposed into individual services consisting of one of many containers. Deploying and maintaining a microservices architecture can be, however, extremely complicated.

A service mesh is a dedicated infrastructure that provides reliable and secure communication between microservices in a distributed system. It consists of proxies that intercept and manage the network traffic between the microservices, and systems responsible for configuring and monitoring those proxies. A service mesh can offer various benefits:

- Load balancing
- Service discovery
- Encryption
- Authentication and authorization
- Observability
- Fault tolerance
- Resilience

OpenShift Service Mesh is a Red Hat OpenShift operator that provides service mesh functionality. It is based on the Istio project, an open-source platform that simplifies the deployment and operation of distributed systems. Be mindful that you cannot use the `istioctl` tool in OpenShift to interact with Istio's control plane components. To learn more about Istio, visit [its official website](#).

Istio also integrates with Kiali, a management console for Service Mesh. Kiali provides a graphical UI that allows you to visualize your mesh topology and health, as well as perform various actions on your services and workloads. You can use Kiali to view metrics, traces, logs, alerts, configurations, validations, and more. To learn more about Kiali, visit [its official website](#).

14.1. Installation

Let's install OpenShift Service Mesh using the operator provided by Red Hat:

1. Navigate to Operators → OperatorHub from the left navigation menu.
2. Search for "OpenShift Service Mesh" in the search box and click on it.
3. Click on "Install" from the pop-up window.
4. Select the default options.
5. Click on Install again from the bottom-right corner.
6. Wait for the operator installation to complete.

Repeat the same operation but for the "Kiali Operator" provided by Red Hat, as well as the "Red Hat

OpenShift distributed tracing platform" operator, also provided by Red Hat.

Istio integrates itself transparently onto existing distributed applications through a proxy sidecar injected into every service deployed as a sidecar container, intercepting and directing traffic between services.

14.2. Using Istio and Kiali

To deploy your applications using Istio's sidecar injection feature, you need to modify your deployments with the `sidecar.istio.io/inject: 'true'` annotation in your deployments. This will instruct Istio to inject Envoy proxies into any pod that is created in those projects.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apitest
spec:
  replicas: 1
  selector:
    matchLabels:
      app: apitest
  template:
    metadata:
      annotations:
        sidecar.istio.io/inject: 'true' ①
      labels:
        app: apitest
    spec:
      volumes:
        # ...
```

① Annotation required to inject the sidecar proxy in your application pod.

Once you have installed Istio and deployed your applications with Envoy proxies, you can start exploring Istio's features and capabilities using Kiali.

Deploy your application and use it for a while. You will see that your pods have two containers now instead of the usual one. Open the "istio-system" project in the OpenShift Web Console and select "Networking → Routes." Open the "Kiali" route, and select your project. You should see now a graphical representation of the interactions between the various components of your microservices architecture.

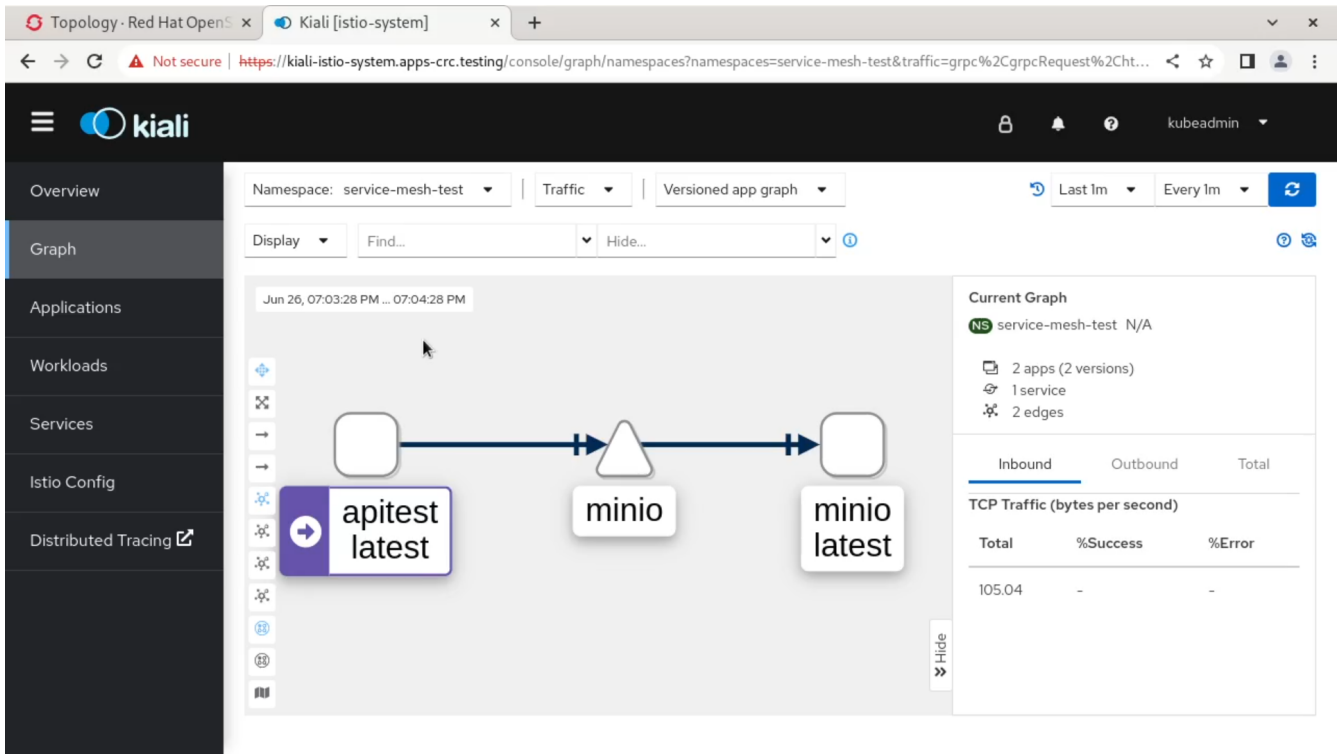


Figure 8. Kiali showing routing between microservices

Scaling and Monitoring Apps

In this last section you'll learn common strategies to monitor the health of your applications on Red Hat OpenShift.

Chapter 15. Logging



The source code for this section is contained in the [04_01 branch of the GitHub repository](#) for this course.

Once your applications are up and running, it is essential to keep an eye on them. The good news is that OpenShift makes it trivial to monitor the behavior of your Cloud Native applications at runtime. OpenShift comes bundled with various open-source and standard tools such as Prometheus and Kibana. No need to install them separately.

Let us talk about logging. Following Kubernetes' best practices and the Twelve-Factor App Principles, your Cloud Native applications should be publishing their log messages as a stream of data directly sent to standard output, so that standard Kubernetes tools can process it. Your application events should be sent one event per line, directly to the "stdout" of your process.

15.1. Checking Pod Logs on the Web Console

The easiest way to check the logs of your pods is to use the Web Console. In the Administration perspective, select the Workloads / Pods menu, and click on any pod running in your cluster. The "Logs" tab will immediately show the current logs of your pod, which will give you a precise idea of the current status of your application.

Checking for logs individually quickly becomes cumbersome, particularly for large microservice applications with many individual components. In particular, Kubernetes can kill pods at any time depending on their runtime status and the current cluster conditions, which means that log messages of a particular pod can be lost at any time.

15.2. Using Elasticsearch and Kibana

Given the nature of Kubernetes deployments, it is better to coalesce application logs using [ElasticSearch](#) and [Kibana](#), two tools that have become the standard Cloud Native way to manage logs. ElasticSearch and Kibana are provided as standard OpenShift operators.



If you are using OpenShift Local, please make sure to configure your CRC cluster to enable monitoring. You must delete any current instance using `crc delete` and then run the `crc config set enable-cluster-monitoring true` command. Then start a new instance with enough disk space and memory: `crc start --memory 20480 --cpus 6 --disk-size 300 --nameserver 1.1.1.1 --pull-secret-file ./pull-secret`

Using the web console:

- Install "OpenShift Elasticsearch Operator" with the defaults.
- Install the "Red Hat OpenShift Logging" operator with the defaults.
- Wait until both operators show the "Succeeded" status in the Operators / Installed Operators screen.
- On the terminal, `oc login` as kubeadmin.

- Apply the logging/cluster-logging-instance.yaml file to your cluster using the `oc` tool, to create an instance of Kibana.

Depending on the amount of RAM in your computer, this process should be faster or slower.

Once all the pods in the "openshift-logging" namespace are ready and running, you should be able to open the menu "applications" on the top-right of the OpenShift web console, and Kibana will open up in a new browser window. You can login to Kibana using the standard username and passwords provided by OpenShift Local at startup. You can define an index pattern and then customize your dashboard with graphs and tables.

Create a new project, deploy the registry.gitlab.com/akosma/linkedin-learning-simpleapi:latest container in it, and execute the "/unstable" path a few times. Then navigate in the OpenShift web console to the pod, and select the "Logs" tab. Click on the [**View in Kibana**] button and this will open the Kibana application with your application logs in it.

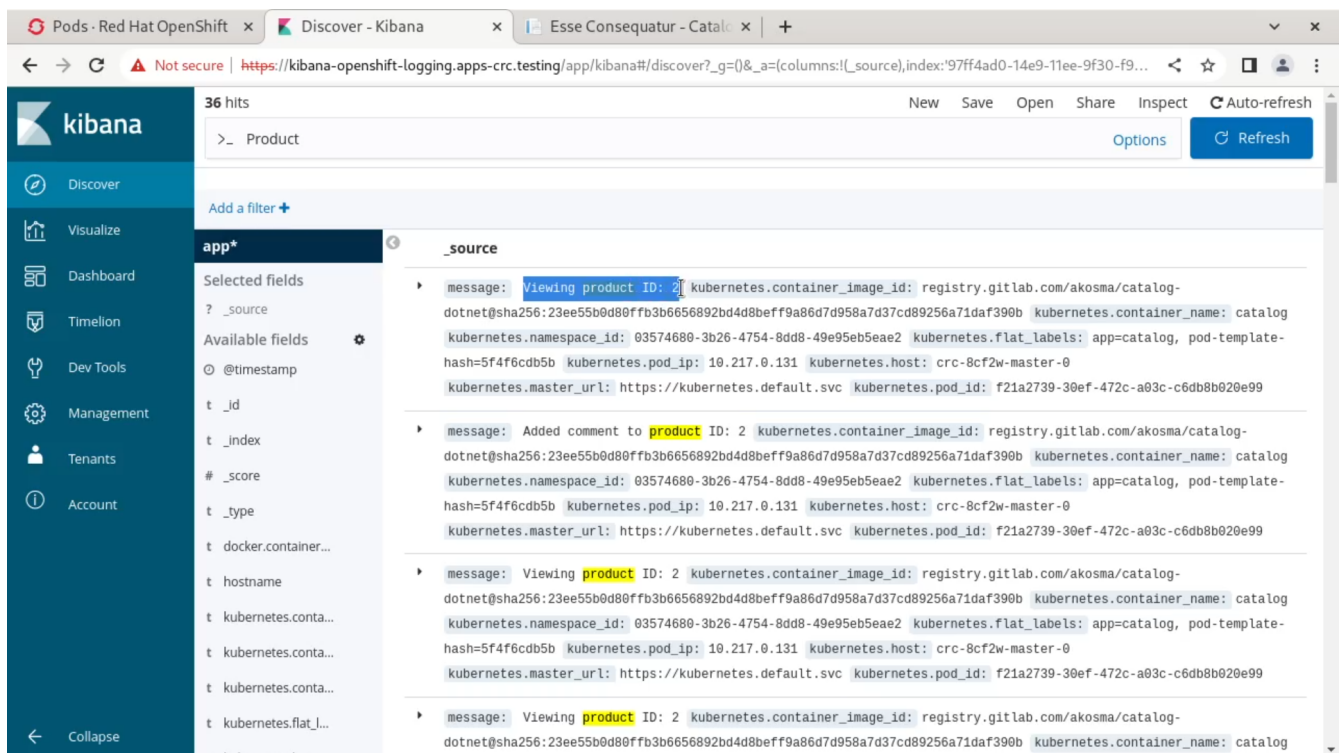


Figure 9. Filtering application logs with Kibana



OpenShift also allows you to forward all logs outside of your cluster, if that's required or desirable.

Chapter 16. Manual, Horizontal, and Vertical scaling



The source code for this section is contained in the [04_02 branch of the GitHub repository](#) for this course.

With OpenShift, you can easily scale your applications to meet changing demands. Scaling can be done in two ways: horizontally or vertically.

Horizontal scaling consists of adding or removing nodes from the cluster, or adding or removing pods from a node. These operations increase or decrease the number of resources available for your applications, such as CPU, memory, disk, and network. Horizontal scaling is useful when you need to handle more traffic, distribute the load, or improve availability and resilience.

Vertical scaling means adjusting the resource limits and requests of the pods that run your applications. This way, you can optimize the resource utilization of your nodes and pods, and ensure that your applications have enough resources to perform well. Vertical scaling is useful when you need to fine-tune the performance, efficiency, or quality of service of your applications.

OpenShift provides several tools and features to help you scale your applications horizontally or vertically. For example:

- You can use a HorizontalPodAutoscaler (HPA) to automatically scale the number of pods based on CPU or memory utilization metrics. The HPA monitors the pods and adjusts the replica count of the workload object (such as Deployment, StatefulSet, etc.) that manages them.
- You can use a VerticalPodAutoscaler (VPA) to automatically adjust the resource limits and requests of the pods based on historical and current resource usage data. The VPA reviews the pods and updates them with optimized resource values before they are admitted to a node.
- You can use a cluster autoscaler (CA) to automatically scale the number of nodes in the cluster based on the demand for resources. The CA monitors the cluster and adds or removes nodes as needed.
- You can use machine sets to manually scale the number of nodes in a specific zone or region. Machine sets are groups of machines that have the same configuration and role.

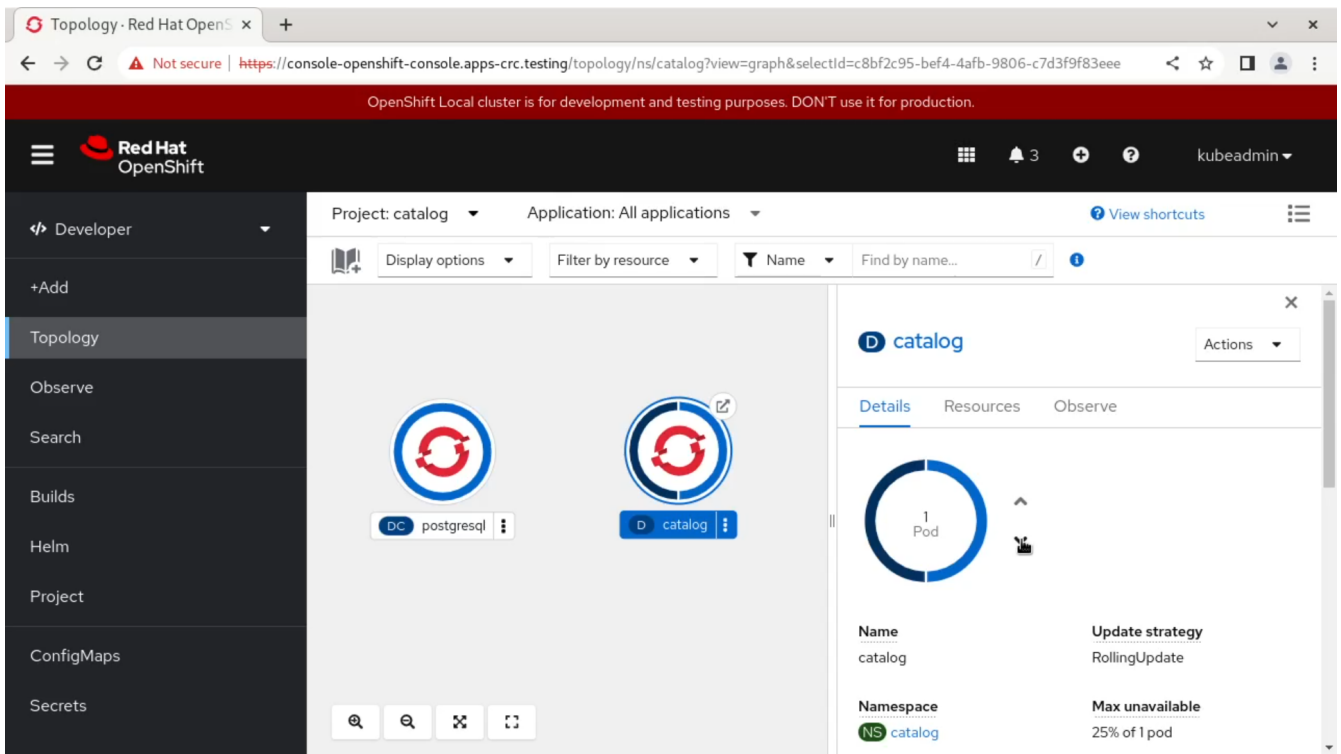


Figure 10. Scaling an application manually through the OpenShift console

Chapter 17. Monitoring apps with Prometheus



The source code for this section is contained in the [04_03 branch of the GitHub repository](#) for this course.

OpenShift includes the Prometheus project, used to observe applications in real time. Software developers can not only keep an eye on the memory and disk space used by their applications, but Prometheus makes it trivial to add counters for any kind of metric that makes sense to your applications.

17.1. Instrumenting Applications

The first step in application instrumentation consists in exporting the required metrics to Prometheus. Many programming languages, such as C#, Python, Java, and JavaScript, and web application frameworks like ASP.NET or Quarkus, include Prometheus-compatible libraries that automatically export data in a format compatible with Prometheus. The endpoints used to export that data usually have the `/metrics` URL.

```
package main

import (
    "github.com/prometheus/client_golang/prometheus" ①
    "github.com/prometheus/client_golang/prometheus/promauto"
    "github.com/prometheus/client_golang/prometheus/promhttp"

    "fmt"
    "net/http"
)

var ( ②
    example_counter = promauto.NewCounter(prometheus.CounterOpts{
        Name: "example_counter",
        Help: "An example of a Prometheus counter",
    })
)

func main() {
    r := prometheus.NewRegistry()
    r.MustRegister(example_counter)

    http.HandleFunc("/", Handler)
    http.Handle("/metrics", promhttp.Handler()) ③
    http.ListenAndServe("0.0.0.0:8080", nil)
}

func Handler(w http.ResponseWriter, r *http.Request) {
```

```

example_counter.Inc()
fmt.Println("Handler executing")
fmt.Fprintf(w, "The simplest API ever!")
}

```

- ① Importing the Prometheus client library for the Go programming language.
- ② Creating a new Prometheus counter.
- ③ Exporting the data to Prometheus using the `/metrics` endpoint



The source code of this application is available on the [akosma/simple-go-api](#) project on GitLab.

Prometheus gathers data from your application at regular intervals, providing an almost real-time trace of your application. Developers and system operators can use this information to understand the usage patterns of the application, and react properly to future events.

17.2. Observing Applications

Prometheus is an open source project that includes a query language called PromQL, used to synthesize and analyze information stored in a Prometheus database. The OpenShift web console includes support for entering PromQL queries, showing the current and historical values of any metric used to track the application.

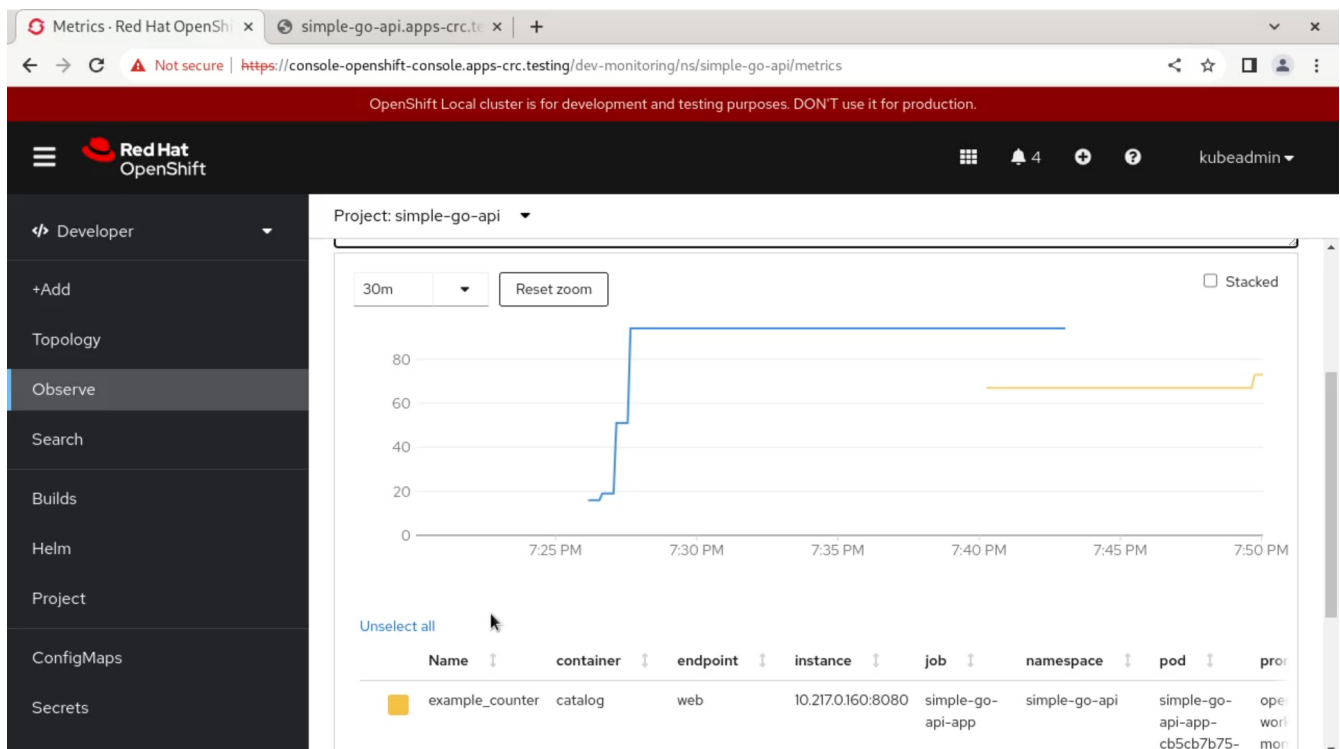


Figure 11. Viewing Prometheus metrics directly on the OpenShift console

Using Grafana

Another common tool used together with Prometheus is [Grafana](#), a web-based visualization tool that natively understands Prometheus data. You can launch an instance of Grafana in your cluster

and show Prometheus data in real time.

Impressum

Last modification: 2023-10-13.

The website openshift.guide has been created and is managed by Adrian Kosmaczewski ([akos.ma](mailto:akos.ma@openshift.guide), info@openshift.guide)

This document, the website openshift.guide and all of its contents are licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. To view a copy of this license, visit creativecommons.org/licenses/by-nc-sa/4.0/.

GDPR Disclaimer

This website does not use cookies. It does not track you. It does not collect any personal information. It does not use Google Analytics. It does not use Facebook Pixel.

It does collect privacy-friendly usage analytics with a self-hosted [Matomo Analytics](#) instance installed in this very same website. Data collection is 100% anonymous and does not require cookies. Check Matomo's [GDPR compliance](#) for more information.

Copyright

Copyright © 2023 Adrian Kosmaczewski. All Rights Reserved. The website openshift.guide nor its author are affiliated with Red Hat, Inc. or any of its subsidiaries in any way.

OpenShift®, Red Hat®, and the Red Hat logo are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries in the United States and other countries. All other trademarks are the property of their respective owners.

Toolchain

This website and associated content in PDF, EPUB, and man page formats have been created using the following tools:

- [Antora](#)
- [Asciidoctor](#)
- Search engine powered by [TNTSearch](#) and [SQLite](#)
- Website built and deployed from a [GitLab](#) CI/CD pipeline.

Index

@

.NET, [14](#), [24](#)

A

Administration perspective, [40](#)

Argo CD, [28](#)

ASP.NET, [44](#)

AWS, [8](#)

AWS Lambda, [34](#)

Azure, [8](#)

Azure Functions, [34](#)

B

BuildConfig, [10](#)

C

C#, [44](#)

C++, [14](#)

CI/CD, [28](#)

Code-Ready Containers, [8](#)

CodeReady Containers, [15](#)

command-line tools, [11](#)

container, [22](#)

container image, [19](#)

container registry, [24](#)

CRC, [8](#), [15](#)

Custom Resource Definitions, [9](#)

D

DeploymentConfig, [9](#)

Developer Catalog, [22](#)

Developer perspective, [22](#), [25](#)

Developer Sandbox for Red Hat OpenShift, [13](#)

E

ElasticSearch, [40](#)

Envoy, [37](#)

F

Flux, [28](#)

G

Git, [25](#)

Git repository, [22](#)

GitHub, [4](#)

Go, [14](#), [24](#)

Google, [34](#)

Google Cloud Platform, [8](#)

Grafana, [45](#)

H

Horizontal scaling, [42](#)

I

IBM Cloud, [8](#)

Importing YAML, [23](#)

install OpenShift Local, [16](#)

J

JAR file, [23](#)

Java, [14](#), [24](#), [44](#)

JavaScript, [24](#), [44](#)

Jenkins, [28](#)

K

Kiali, [36](#)

Kibana, [40](#), [40](#)

Knative, [34](#)

Kubernetes, [7](#)

L

logs, [40](#)

M

Microservices, [36](#)

monitor, [40](#)

N

Node.js, [14](#)

non-root accounts, [20](#)

O

OpenShift 4.12, [33](#)

OpenShift Kubernetes Distribution, [8](#)

OpenShift Service Mesh, [36](#)

operator, [28](#), [36](#)

OperatorHub, [33](#), [36](#)

Operators, [33](#)

P

perspectives, [22](#)
PHP, [14](#), [24](#)
Platform-as-a-Service, [8](#)
privilege escalation, [19](#)
privileged ports, [20](#)
Project, [9](#)
Prometheus, [40](#), [44](#)
PromQL, [45](#)
Python, [14](#), [24](#), [44](#)

Q

Quarkus, [14](#), [44](#)

R

Red Hat developer account, [13](#)
Red Hat OpenShift, [7](#)
Red Hat OpenShift Dev Spaces, [14](#)
Red Hat OpenShift Local, [8](#), [15](#)
Red Hat OpenShift Pipelines, [28](#)
Red Hat Quay, [20](#)
Red Hat Universal Base Images, [19](#)
role, [19](#)
Route, [9](#)
Rust, [14](#)

S

Scala, [14](#)
Scaling, [42](#)
secure by default, [19](#)
Security Context Constraints, [19](#)
Serverless, [34](#)
service mesh, [36](#)
source code project, [22](#)
stateful applications, [33](#)

T

Tekton, [28](#)
templates, [32](#)
Topology, [27](#)
Twelve-Factor App, [21](#), [40](#)
TypeScript, [24](#)

U

UBI, [19](#)

V

Vertical scaling, [42](#)

Visual Studio Code, [14](#)

W

Web Console, [22](#), [40](#)